Machine Learning
Programming Assignment 3

Vaibhav Jindal
111701029

- Leanr forward-backward algorithm with learning HMM including all it’s math and probabilites.
- Learnt the 3 problems in HMM: Evaluation, Decoding and Learning.
- Actually coded full HMM because didn’t know that there is in-build library in scipy for HMM (now I know it).
- Learned the importance of scaling when T (input length) is very large.

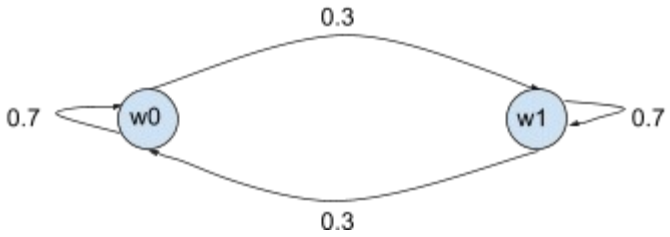
Task 1: Preprocessing: Inspect the (very modest!) "hmm-train". Clean it up by making all letters uppercase, replacing all non-alphabetic characters (numbers, punctuation, and whitespace, including line breaks) with spaces, and then collapsing sequences of spaces to be one space (this should take 2-3 lines of python/matlab code). You will be treating this dataset as a stream of characters (that's why we don't want any line breaks).

- Inserting input data into a string with spaces and uppercase letters
- After that collapsing spaces using join

Task 2: Using your knowledge of language, suggest a "natural" 2-state HMM model with transition and emission probabilities defined by you. Try to think of two meaningful states between which the data as a string of characters is transitioning. You are allowed and encouraged to do any measurement on the training data to optimize the proposed HMM, but you are NOT allowed to use the forward-backward training algorithm learned in class. Please propose any model (and the probabilities) that you think is reasonable, and don't spend too much time to generate the "best" HMM. Draw the two states and show the transition probabilities of your model (similar to the drawing in the slides). Also show the 7 most likely characters in each of the two emission distributions with their probabilities.

Emmision probability same for both hidden states, each same value 1/27

Transition probability, self is 0.7, changing is 0.3



Task 3: Implement the an algorithm for training the two-state HMM model. Print out and inspect the final transition and emission probabilities. Draw the two states and show the transition probabilities and the 7 most likely characters in each of the two emission distributions, with their probabilities. Can you interpret the emergent states?

starting state probability, pie values
[pie[0] = 0.49866048 pie[1] = 0.50133952]]
As can be seen w1 has higher probility of being initial state

hidden states probability
[
[w0->w0 = 0.50114887 w0->w1 = 0.49885113]
[w1->w0 = 0.49914974 w1->w1 = 0.50085026]
]
Hence self state probility is slightly higher for both hidden states

hidden to visible state probabilties
[
w0 to visible states
[6.22334689e-02 1.17917582e-02 1.90288531e-02 3.68574789e-02
1.04187265e-01 2.11086967e-02 1.28470833e-02 5.46056702e-02
5.69043604e-02 8.00674534e-04 3.23818808e-03 3.18081179e-02
2.32257400e-02 5.62495481e-02 6.43805581e-02 1.13154650e-02
1.16460432e-03 5.12427207e-02 5.28807639e-02 6.80202203e-02
2.20544509e-02 1.02966293e-02 1.94709820e-02 7.64057796e-04
1.73238043e-02 3.63833920e-05 1.86162457e-01]

w1 to visible states
[6.22705285e-02 1.17984728e-02 1.90504427e-02 3.68984036e-02
1.04193093e-01 2.11207292e-02 1.28546190e-02 5.46083537e-02

5.69694805e-02 8.01131275e-04 3.24184505e-03 3.18272587e-02
2.32266247e-02 5.63137242e-02 6.44191913e-02 1.13282462e-02
1.16529505e-03 5.12728513e-02 5.29112289e-02 6.80604645e-02
2.20680185e-02 1.03084200e-02 1.94820229e-02 7.64938791e-04
1.73334486e-02 3.64259695e-05 1.85674741e-01]
]

- As it can be seen that from both the states probility of space ‘ ’ (index 26), given wi is maximum. Probability = 0.18616 in w0 and 0.18567 in w1
- Second largest probability is of ‘e’ (index 4). Probability = 0.104187 in w0 and 0.104193 in w1

Task 4: Compare the model of the "natural" HMM you proposed in (2) with the HMMs trained by the code(3). Does the natural HMM perform better than the trained HMMs? Now initialise the HMM to this natural solution, and then run the training algorithm from this starting point. Can this degrade the natural solution? Did you succeed in improving on the solution in? Is there any change in training time ? What do you learn from that ?

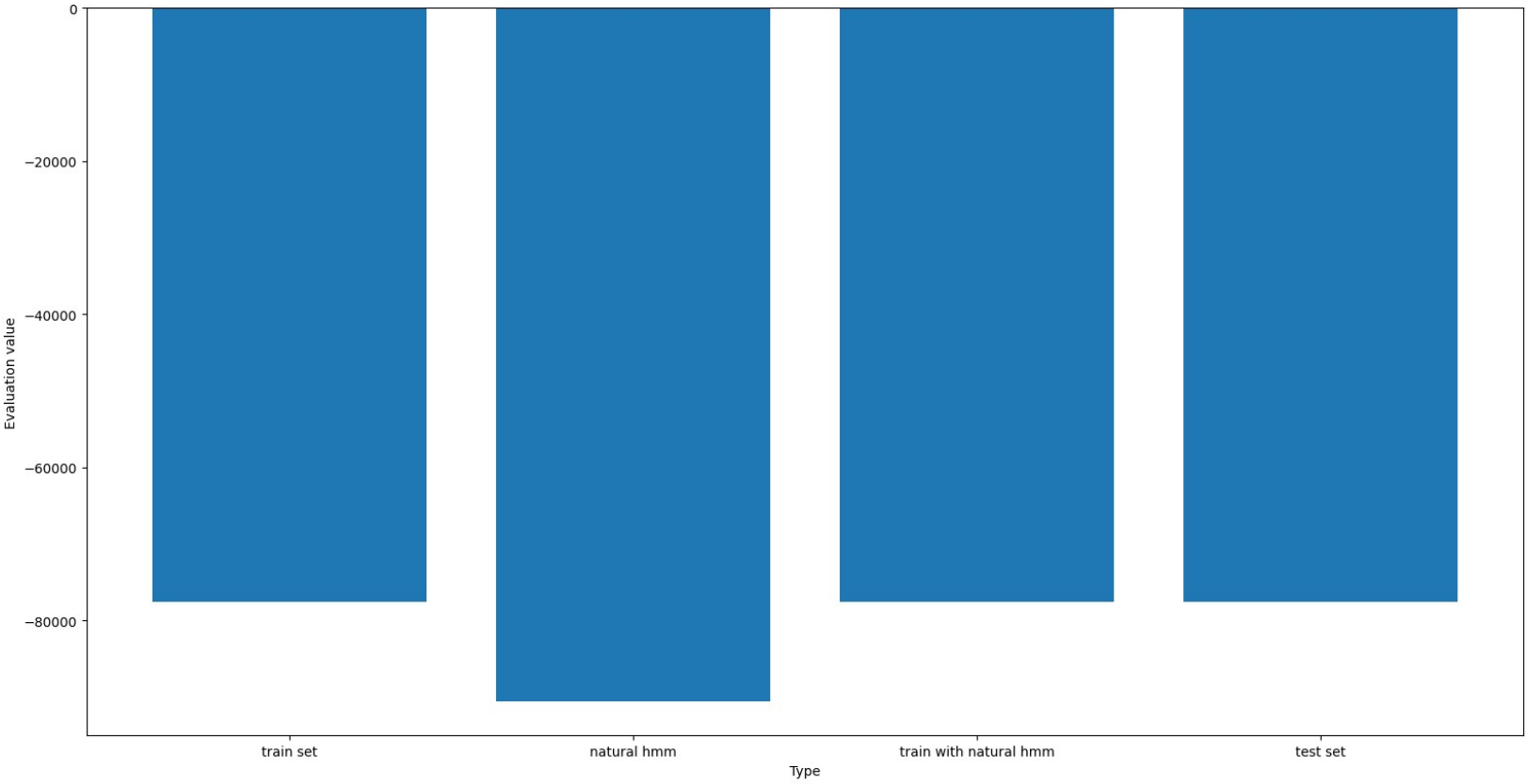
Evaluation with given theta
log[P(O | λ)] Log Probability of training set by code(3) = -77556.61023272702

Evaluation ‘natural hmm’ with given theta
log[P(O | λ)] Log Probability of natural hmm without learning = -90533.3428722604

Evaluation of hmm with initialization with natural hmm
log[P(O | λ)] Log Probability of training set after learning with natural hmm initialization -77556.61022881964

Points of observation:

- It can be easily seen that natural hmm is very bad without learning and has the minum value.
- Hmm with random initialization is medium.
- Hmm with natural solution as initialization is the best one with best log[P(O | λ)], although it is very near to the random initialization one.
- Time taken by the natural solution initialization is somewhat less as compared to the random initialization one.
- Without scaling using c values (please refer the code), all the alpha and beta values quickly vanishes to 0, it is due to big T.



Task 5: Using your best solution, evaluate this test set (after cleaning it up). Compare it to the training set. Is there significant overfitting?

Comparision can be seen in above figure.

Evaluation on test set
log[P(O | λ)] Log Probability of test hmm -77559.9103712009

Note:

- Evaluation on test set is less that the training set.
- But the value is very near, hence cannot be considered as overfitting.

Conclusion: Hmm is a very slow algorithm. Also as it see all the possible paths final probability P(O | λ) comes out to be very small.